UE5 Performance Monitor System (UE5-PMS)

Technical Documentation

What is UE5-PMS?

UE5 Performance Monitor System (UE5-PMS) is a C++/Blueprint plugin for Unreal Engine 5 that lets you:

- Query engine-level performance stats via RHI (platform-agnostic path).
- Query vendor-specific GPU metrics on NVIDIA GPUs via NVAPI.
- Expose both sets of data directly to Blueprints.
- Display the data in in-game debug overlays, dev HUDs, or custom tools.

The goal is to give developers fine-grained performance insight inside their game without external tools, while staying lightweight and Blueprint-friendly.

Key Features

- RHI Stats Node
- Access core engine timing stats (frame time, game/render thread time, RHI timing, etc.).
- Works on all GPUs supported by Unreal's RHI (platform-agnostic).
- Designed for shipping builds as well as PIE.
- NVIDIA GPU Node
- Uses NVAPI to read GPU utilization, VRAM usage, temperature, clocks, and power (where supported).
- NVIDIA-only; automatically reports "unavailable" / default values when unsupported.
- Blueprint-Friendly
- Clean Blueprint nodes with struct outputs.
- Easy integration into existing HUD widgets or debug UIs.
- Runtime Safe
- Designed to run in packaged builds.
- Graceful fallback if vendor-specific metrics aren't available.

Requirements & Installation
Engine Version
- Built and tested for Unreal Engine 5.x (update your exact tested range here, e.g. 5.3–5.4).
- Uses standard RHI stats and C++ plugin workflow.
Platform & Hardware
- RHI Stats Node
- Works on any platform / GPU where Unreal's RHI provides stats.
- NVIDIA GPU Node
- NVIDIA GPU required.
- NVAPI must be available on the target machine (standard for modern NVIDIA drivers on Windows).
Non-NVIDIA GPUs can still fully use the RHI Stats Node; the NVIDIA node will simply not report vendor metrics.
Installation
1. Copy Plugin Folder
- Place the UE5PMS folder into:
- YourProject/Plugins/ (project plugin), or
- Engine/Plugins/ (engine-wide).
2. Regenerate Project Files
- Right-click your .uproject → Generate Visual Studio project files (or equivalent for your IDE).
3. Compile
- Open the project in your IDE and build, or let Unreal compile on first launch.
4. Enable Plugin
- Open Unreal → Edit → Plugins → search for UE5-PMS.
- Enable it and restart the editor if prompted.
Data Sources & Architecture
UE5-PMS exposes performance data through two main paths:
1. RHI Stats Path (Engine-Level)
- Uses Unreal's RHI and stat system to read:
- Frame time
- Game thread time
- Render thread time
- RHI thread / GPU time (where supported)
- Other engine-level timing metrics
- Vendor-agnostic and safe for all GPUs.
2. NVIDIA GPU Path (Vendor-Specific)
- Uses NVAPI to query:
- GPU core utilization
- Memory controller utilization
- VRAM usage
- GPU temperature (°C)
- Core clock / memory clock
- Power usage and power limit (% where supported)
- Only active when an NVIDIA GPU and NVAPI are available.

High-Level Flow

1. Blueprint calls a UE5-PMS node (RHI or NVIDIA).
2. C++ backend reads from:
- Engine stats (RHI).
- NVAPI (vendor metrics).
3. Data is packed into struct outputs.
4. Blueprint uses those structs to:
- Update a debug overlay.
- Log performance.
- Trigger dev tools / profiling logic.

Blueprint Overview

UE5-PMS currently ships with two primary Blueprint nodes:

1. Get RHI Performance Stats
2. Get NVIDIA GPU Stats

Each node returns a struct containing all relevant metrics. You can break the struct to access individual fields.

Common Usage Pattern

1. Create a Blueprint Function Library or a Manager Actor/Subsystem.
2. On a Timer, Tick, or dev-toggle:
- Call Get RHI Performance Stats.
- Optionally call Get NVIDIA GPU Stats (if you care about vendor metrics).
3. Store the latest data in variables and use it in:
- UMG widget (overlay).
- Debug log / on-screen messages.
- Custom monitoring systems.

RHI Stats Node

Node: Get RHI Performance Stats

Purpose: Fetch core engine performance data via the RHI/Stats system.

Recommended Output Struct Fields (example; match to your implementation):
- FrameTimeMs – Total frame time in milliseconds.
- GameThreadTimeMs – Time spent on the game thread.
- RenderThreadTimeMs – Time spent on the render thread.
- RHIThreadTimeMs / GPUTimeMs – RHI or GPU time where available.
- DeltaTimeSeconds – Fixed or variable delta time from the engine.
- FramesPerSecond – Derived FPS.

You can also include additional fields like:
- NumPrimitives, NumDrawCalls, etc. (if you're exposing them).
- Any other stat group values you've wired into the struct.

Example Blueprint Usage
- Create a Dev Performance Overlay widget:
- Bind text fields to variables like CurrentFrameTimeMs, CurrentGPUTimeMs, etc.
- In your Player Controller or Game Instance:
- On a timer (e.g. every 0.25s) call Get RHI Performance Stats.
- Update the bound variables.
- Result: a live performance readout that works across all GPUs.

Behavior in Shipping Builds
- The node is designed to function in packaged builds.
- It relies on engine stats that are available at runtime; where a stat is disabled or not supported, the corresponding value may be 0 or a default.

NVIDIA GPU Stats Node
Node: Get NVIDIA GPU Stats
Purpose: Provide access to NVIDIA-specific GPU metrics via NVAPI for deeper profiling.

Typical Output Struct Fields (example):
- bIsNvidiaGPU – Boolean flag if an NVIDIA GPU is detected.
- bIsDataValid – Whether NVAPI returned valid readings this frame.

Core Metrics:
- GPUUtilizationPercent – Overall GPU workload.
- MemoryUtilizationPercent – VRAM controller utilization.
- VramUsedMB – Estimated VRAM usage in MB.
- VramTotalMB – Total VRAM in MB (if available).
- CoreClockMHz – Current GPU core clock.
- MemoryClockMHz – Current memory clock.
- TemperatureCelsius – Core temperature.
- PowerUsagePercent – Power draw relative to TDP (if reported).

Fallback Behavior

If the node is called on a system where:
- No NVIDIA GPU is present, or
- NVAPI is unavailable or fails,

then:
- bIsNvidiaGPU and/or bIsDataValid will be false.
- Numeric fields will return 0 or safe defaults.

You can branch off bIsDataValid in Blueprint to hide NVIDIA-only sections in your overlays.

Creating a Performance Overlay UI
1. Dev Manager Setup
- Create a Blueprint BP_PerformanceMonitorManager (or similar).
- Add variables:
- RHIStats (struct from RHI node).
- NvidiaStats (struct from NVIDIA node).
- Add a RefreshInterval (e.g. 0.25 seconds).
- On BeginPlay:
- Set a Timer by Event with RefreshInterval.
- Each tick of the timer:
- Call Get RHI Performance Stats → store in RHIStats.
- Call Get NVIDIA GPU Stats → store in NvidiaStats.
2. Overlay Widget
- Create WBP_PerformanceOverlay.
- Expose RHIStats and NvidiaStats as Expose on Spawn or via setters.
- Add text blocks for:
- FPS, FrameTime, GameThreadTime, RenderThreadTime.
- GPU Util, VRAM, Temp, etc. (only show if bIsDataValid).
- In Tick or via Bindings:
- Pull from RHIStats and NvidiaStats to update displayed values.
3. Enabling / Disabling Overlay
- Add a keybinding (e.g. F1) to toggle:
- Add to viewport / remove from parent.
- Optionally gate it behind a console variable or dev-only setting.

Best Practices & Integration Tips
Don’t Over-Poll
- Polling every frame is possible but not necessary.
- Suggested intervals:
- 0.1–0.5 seconds for a smooth yet stable readout.
- Longer intervals for logging or telemetry.
Separate Dev & Shipping UI
- Keep heavy debug overlays dev-only.
- For shipping builds, you can:
- Provide a slim, low-cost overlay behind a dev toggle.
- Log metrics to files or send to your own telemetry back-end.
Use RHI as the Baseline
- For cross-hardware compatibility:
- Always rely on RHI stats as your baseline.
- Treat vendor-specific metrics as bonus detail when available.
Error Handling
- Always branch on bIsDataValid (or equivalent) for vendor-specific nodes.
- Avoid assuming a metric is non-zero; handle missing or unsupported fields gracefully.
Troubleshooting & FAQ
Q: All my RHI stats are showing 0, what’s wrong?
- Ensure:
- The plugin is enabled and compiled correctly.
- You’re running in a context where stats are available.
- Some platforms or configurations might restrict certain stat groups; where a stat is disabled, its value may remain at 0.

Q: NVIDIA stats are all zero / invalid.
- Check:
- There is an NVIDIA GPU present.
- Drivers are installed and up to date.
- In Blueprint:
- Verify bIsDataValid (or equivalent) is true.
- If false, the plugin has gracefully failed to read vendor metrics and you should hide or grey-out those UI elements.

Q: Does this work in packaged builds?
- Yes. Both RHI and NVIDIA nodes are designed to work in shipping builds.
- Some debugging or extra logging you add yourself may be editor-only; keep core node calls runtime-safe.

Q: Can I log this to a file or send it to a server?
- Yes. The plugin exposes data via Blueprint structs.
- You can:
- Write to CSV/JSON via existing file I/O plugins.
- Forward metrics to your own telemetry system.

Licensing & Third-Party Notices

Plugin License

(Insert your chosen license text here, e.g. MIT / proprietary terms, etc.)

Third-Party Components

UE5-PMS depends on:
- NVIDIA NVAPI
- Used solely to query NVIDIA GPU metrics where available.
- For details, refer to NVAPI’s official license and documentation (linked or summarized in your separate ThirdParty notice file).

All other functionality is built on top of Unreal Engine’s standard APIs and RHI.